

# ... TAREA 1 ...

Flujo de trabajo con Github

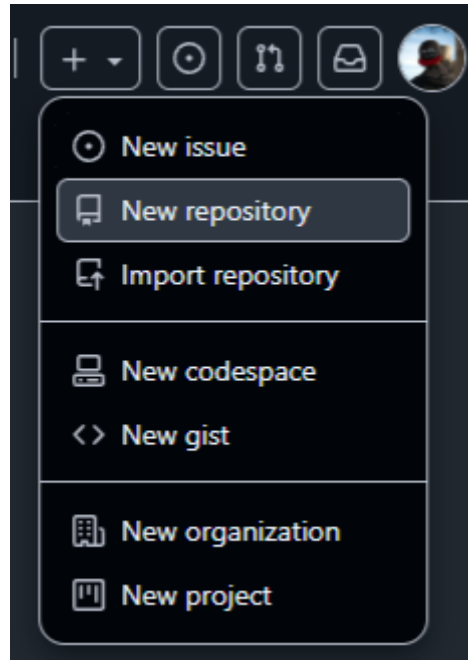


<b>CREANDO Y CLONANDO EL REPOSITORIO.....</b>	<b>3</b>
CREAR REPOSITORIO GITHUB.....	3
CLONANDO REPOSITORIO GITHUB.....	5
<b>CREANDO EL “README.me”, “.gitignore” Y LAS RAMAS.....</b>	<b>5</b>
CREANDO README.me.....	5
CREANDO RAMAS ADICIONALES.....	7
CREANDO .gitignore.....	8

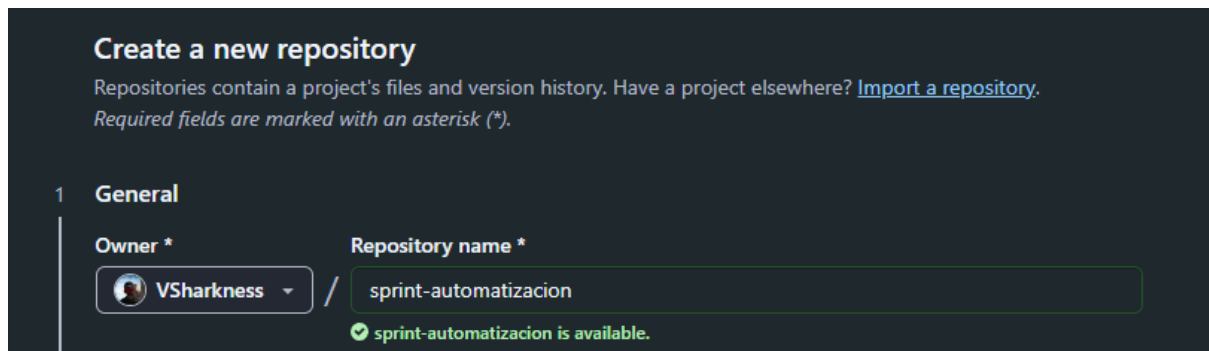
# CREANDO Y CLONANDO EL REPOSITORIO

## CREAR REPOSITORIO GITHUB

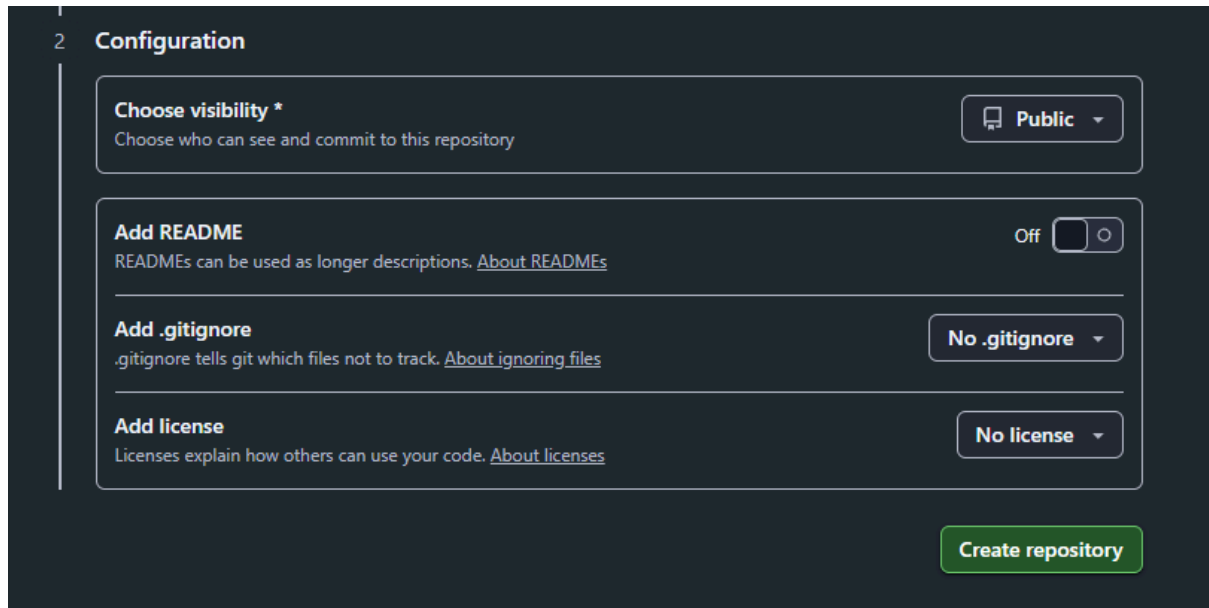
Primero, desde <https://github.com/>, accedemos al icono del símbolo “+” y pulsamos sobre “New repository”.



Aquí debemos elegir el nombre de nuestro repositorio, en este caso “sprint-automatizacion”.

A screenshot of the 'Create a new repository' form on GitHub. The form has a dark background. At the top, it says 'Create a new repository' in bold, followed by a description: 'Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#) Required fields are marked with an asterisk (\*).' Below this, there is a section titled '1 General'. Under 'General', there are two main fields: 'Owner \*' and 'Repository name \*'. The 'Owner \*' field has a dropdown menu showing 'VSharkness' with a profile picture icon. The 'Repository name \*' field is a text input containing 'sprint-automatizacion'. Below the 'Repository name' field, there is a green checkmark icon followed by the text 'sprint-automatizacion is available.'

Nos permitirá elegir si queremos hacerlo Público y, opcionalmente, podemos crear el “README.me” y el “.gitignore” desde aquí. Por ahora no lo haremos, ya que los agregaremos más tarde mediante Git.

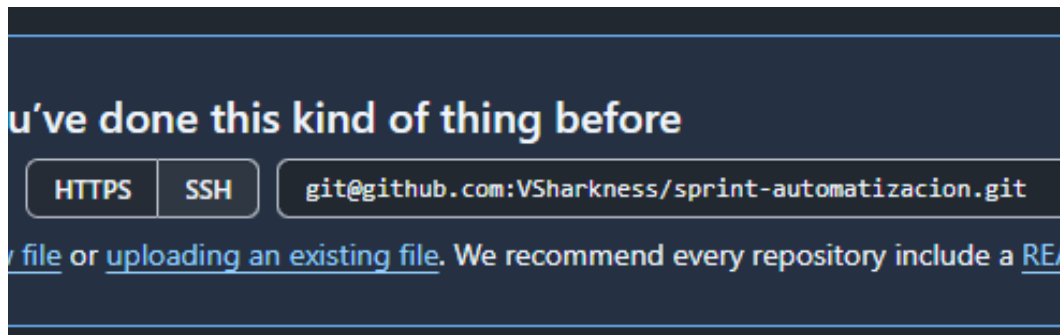


The screenshot shows the '2 Configuration' step of creating a new repository on GitHub. It features three main configuration sections, each with a dropdown menu on the right:

- Choose visibility \***: A dropdown menu set to 'Public'. Below it, a link says 'Choose who can see and commit to this repository'.
- Add README**: A toggle switch set to 'Off'. Below it, a link says 'READMEs can be used as longer descriptions. [About READMEs](#)'.
- Add .gitignore**: A dropdown menu set to 'No .gitignore'. Below it, a link says '.gitignore tells git which files not to track. [About ignoring files](#)'.
- Add license**: A dropdown menu set to 'No license'. Below it, a link says 'Licenses explain how others can use your code. [About licenses](#)'.

At the bottom right, there is a green button labeled 'Create repository'.

Una vez creado, nos mostrará un enlace con la **dirección HTTPS o SSH**. En nuestro caso, guardaremos la dirección SSH.



The screenshot shows the top part of a newly created repository page on GitHub. It has a dark blue header with the text 'You've done this kind of thing before'. Below this, there are two buttons: 'HTTPS' and 'SSH'. The 'SSH' button is selected, and the URL 'git@github.com:VSharkness/sprint-automatizacion.git' is displayed in a light blue box. Below the URL, there is a link that says 'clone or uploading an existing file. We recommend every repository include a README file'.

## CLONANDO REPOSITORIO GITHUB

Para clonar el repositorio basta con abrir la Terminal e ingresar mediante el comando “**cd**” a la carpeta donde queramos clonar el repositorio. Una vez ahí, utilizamos “**git clone *direcciondelrepositorio.git***” para clonar el repositorio en esa carpeta.

*\*La dirección del repositorio es la que recibimos en el paso anterior.*

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint$ git clone git@github.com:VSharkn
ess/sprint-automatizacion.git
Clonando en 'sprint-automatizacion'...
warning: Parece haber clonado un repositorio sin contenido.
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint$
```

Comprobamos que la carpeta se ha creado mediante el comando “**ls**” y accedemos a ella mediante el comando “**cd**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint$ ls
sprint-automatizacion
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint$ cd sprint-automatizacion
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$
```

## CREANDO EL “README.me”, “.gitignore” Y LAS RAMAS

### CREANDO README.me

Creamos un archivo con texto incluido mediante el comando “**echo**” y lo llamamos “**README.me**”, agregándole el texto “**Sprint**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ echo "Sprint" > README.md
```

Preparamos los cambios mediante el comando “**git add README.md**”

Además, mediante el comando “**git status**” comprobamos que se está subiendo correctamente en la rama master.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git add README.md
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git status
En la rama master

No hay commits todavía

Cambios a ser confirmados:
  (usa "git rm --cached <archivo>..." para sacar del área de stage)
nuevos archivos: README.md
```

Creamos el commit mediante “**git commit -m “Descripción del commit”**”.

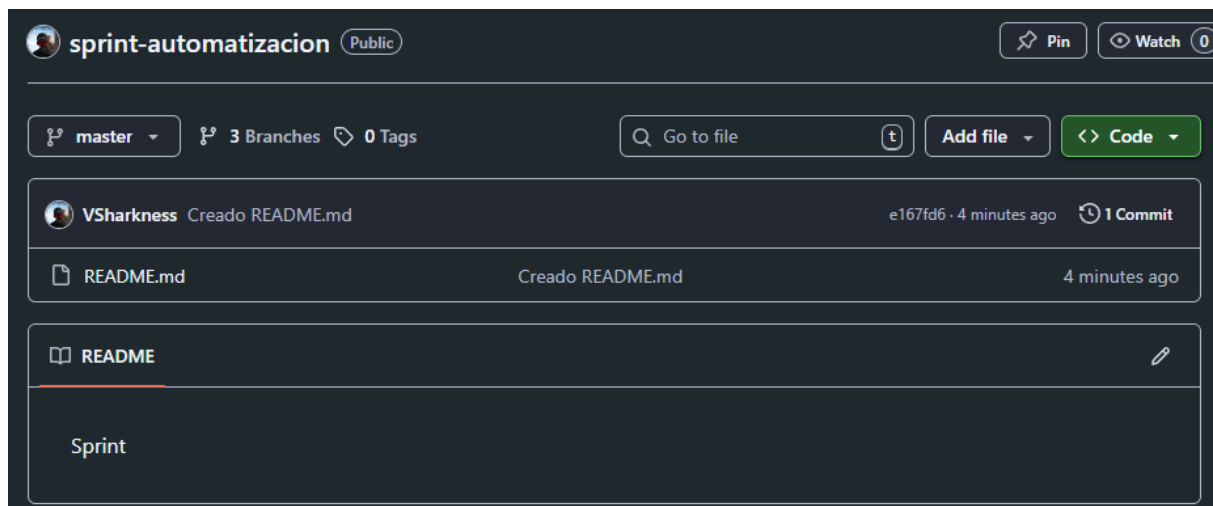
```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git commit -m "Creado README.md"

[master (commit-raiz) e167fd6] Creado README.md
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

Y por último lo enviamos al repositorio remoto en la rama master (vinculando mediante -u para así establecer un vínculo entre la rama local y la remota) mediante el comando “**git push -u origin master**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git push -u origin master
Enumerando objetos: 3, listo.
Contando objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 229 bytes | 229.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To github.com:VSharkness/sprint-automatizacion.git
 * [new branch]      master -> master
rama 'master' configurada para rastrear 'origin/master'.
```

Comprobamos desde nuestro repositorio remoto que el archivo se ha enviado correctamente:



## CREANDO RAMAS ADICIONALES

Crearemos la rama dev mediante el comando “**git checkout -b dev**”. Agregamos el **-b** para que cree y te cambie a la nueva rama.

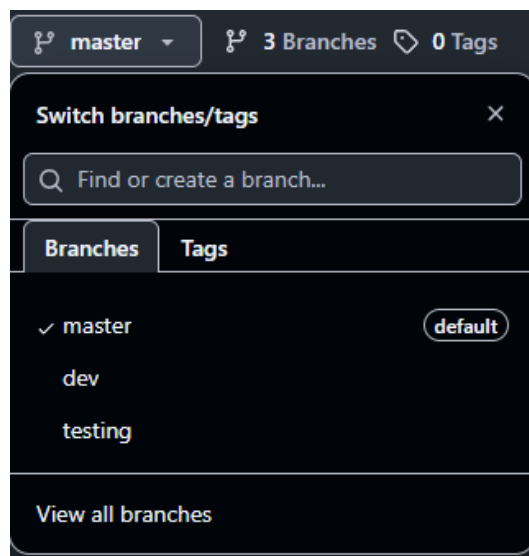
Tras eso, mediante el comando “**git push -u origin dev**” subimos la rama al repositorio remoto, además de vincular la rama remota dev con la local del mismo nombre.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git checkout -b dev
Cambiado a nueva rama 'dev'
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git push -u origin dev
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/VSharkness/sprint-automatizacion/pull/new/dev
remote:
To github.com:VSharkness/sprint-automatizacion.git
* [new branch]      dev -> dev
rama 'dev' configurada para rastrear 'origin/dev'.
```

Hacemos exactamente el mismo paso para la rama “**testing**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git checkout -b testing
Cambiado a nueva rama 'testing'
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git push -u origin testing
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
remote:
remote: Create a pull request for 'testing' on GitHub by visiting:
remote:   https://github.com/VSharkness/sprint-automatizacion/pull/new/testing
remote:
To github.com:VSharkness/sprint-automatizacion.git
* [new branch]      testing -> testing
rama 'testing' configurada para rastrear 'origin/testing'.
```

En el repositorio remoto, comprobamos que ambas ramas se han enviado correctamente:



## CREANDO .gitignore

Mediante el comando “**nano .gitignore**” abrimos el editor nano para indicar que archivos o carpetas queremos que git ignore.

```
GNU nano 7.2
# Python
__pycache__/
*.pyc
*.pyo
*.pyd
.env
venv/
*.env
# Bash
*.log
*.tmp
```

- **\_\_pycache\_\_**/: Carpeta que utiliza Python para los archivos compilados.
- **\*.pyc, \*.pyo y \*.pyd**: Archivos compilados de Python.
- **.env**: Variables de entorno
- **venv/**: Entorno virtual de Python.
- **\*.env**: Otros posibles archivos de configuración de entorno.
- **\*.log**: Archivos de registro creados por scripts de Bash.
- **\*.tmp**: Archivos temporales que pueden crearse en Bash.

Como hicimos con el README.md, utilizaremos el comando “**add**” para añadir el **.gitignore** y “**status**” para comprobar que está correctamente añadido desde la rama master. Para futuros archivos utilizaremos las otras ramas, pero para estos archivos principales pueden subirse en la master.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git add .gitignore
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
nuevos archivos: .gitignore
```

Creamos el commit con “**git commit -m “Descripción del commit”**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git commit -m "Creado .gitignore para Python y Bash"
[master d183028] Creado .gitignore para Python y Bash
1 file changed, 11 insertions(+)
create mode 100644 .gitignore
```



Por último, enviamos todos los nuevos cambios a la rama master mediante el comando “**git push origin master**”.

```
marcos@VB-Ubuntu:~/Escritorio/Prácticas/Sprint/sprint-automatizacion$ git push origin master
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 363 bytes | 30.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To github.com:VSharkness/sprint-automatizacion.git
e167fd6..d183028  master -> master
```

Podemos comprobar como están tanto las ramas como los archivos correctamente en repositorio:

